The tractability of CSP classes defined by forbidden patterns

The tractability of CSP classes defined by forbidden patterns

David A. Cohen
Royal Holloway, University of London,
dave@cs.rhul.ac.uk

Martin C. Cooper
IRIT, University of Toulouse III,
cooper@irit.fr

Páidí Creed
Royal Holloway, University of London,
paidi@cs.rhul.ac.uk

András Z. Salamon
University of Edinburgh,
asalamon@inf.ed.ac.uk

July 9, 2014

Abstract

The constraint satisfaction problem (CSP) is a general problem central to computer science and artificial intelligence. Although the CSP is NP-hard in general, considerable effort has been spent on identifying tractable subclasses. The main two approaches consider *structural* properties (restrictions on the hypergraph of constraint scopes) and *relational* properties (restrictions on the language of constraint relations). Recently, some authors have considered *hybrid* properties that restrict the constraint hypergraph and the relations simultaneously.

Our key contribution is the novel concept of a *CSP pattern* and classes of problems defined by *forbidden patterns* (which can be viewed as forbidding generic sub-problems). We describe the theoretical framework which can be used to reason about classes of problems defined by forbidden patterns. We show that this framework generalises relational properties and allows us to capture known hybrid tractable classes.

Although we are not close to obtaining a dichotomy concerning the tractability of general forbidden patterns, we are able to make some progress in a special case: classes of problems that arise when we can only forbid binary negative patterns (generic sub-problems in which only inconsistent tuples are specified). In this case we are able to characterise very large classes of tractable and NP-hard forbidden patterns. This leaves the complexity of just one case unresolved and we conjecture that this last case is tractable.

Keywords: Constraint satisfaction problem, tractability, forbidden substructures.

1 Introduction

In the constraint satisfaction paradigm we consider computational problems in which we have to assign values (from a *domain*) to *variables*, under some *constraints*. Each constraint limits the (simultaneous) values that a list of variables (its *scope*) can be assigned. In a typical situation some pair of variables might represent the starting times of two jobs in a machine shop scheduling problem. A reasonable constraint would require a minimum time gap between the values assigned to these two variables.

Constraint satisfaction has proved to be a useful modelling tool in a variety of contexts, such as scheduling, timetabling, planning, bio-informatics and computer vision. This has led to the development of a number of successful constraint solvers. Unfortunately, solving general constraint satisfaction problem (CSP) instances is NP-hard and so there has been significant research effort into finding tractable fragments of the CSP.

In principle we can stratify the CSP in two quite distinct and natural ways. The structure of the constraint scopes of an instance of the CSP can be thought of as a hypergraph where the variables are the vertices, or more generally as a relational structure. We can find tractable classes by restricting this relational structure, while allowing arbitrary constraints on the resulting scopes [10]. Sub-problems of the general constraint problem obtained by such restrictions are called structural. Alternatively, the set of allowed assignments to the variables in the scope can be seen as a relation. We can choose to allow only specified kinds of constraint relations, but allow these to interact in an arbitrary structure [20]. Such restrictions are called relational or language-based.

Structural subclasses are defined by specifying a set of hypergraphs (or relational structures) which are the allowed structures for CSP instances. It has been shown that tractable structural classes are characterised by limiting appropriate (structural) width measures [11, 13, 19, 15, 23]. For example, a tractable structural class of binary CSPs is obtained whenever we restrict the constraint structure (which is a graph in this case) to have bounded tree width [11, 13]. In fact, it has been shown that, subject to certain complexity-theoretic assumptions, the only structures which give rise to tractable CSPs are those with bounded (hyper-)tree width [9, 17, 18, 23].

Relational subclasses are defined by specifying a set of constraint relations. It has been shown that the complexity of the subclass arising from any such restriction is precisely determined by the so called *polymorphisms* of the set of relations [1, 4]. The polymorphisms specify that, whenever some set of tuples is in a constraint relation, then it cannot be the case that a particular tuple (the result of applying the polymorphism) is not in the constraint relation. It is thus the relationship between allowed tuples and disallowed tuples inside the constraint relations that is of key importance to the relational tractability of any given class of instances. Whilst a general dichotomy has not yet been proven for the relational case, many dichotomies on sub-problems have been obtained, e.g. [2, 1, 3].

Unfortunately, by allowing only structural or relational restrictions we limit the possible subclasses that we can define. By considering restrictions on both the structure of the constraint graph and the relations, we are able to identify new tractable classes. We call these restrictions *hybrid* reasons for tractability.

Several hybrid results have been published for binary CSPs [22, 5, 28, 6, 7]. Instead of looking at the constraint graph or the constraint language, these works captured tractability based upon the properties of the (coloured) microstructure of CSP instances. The *microstructure* of a binary CSP instance is the graph $\langle V, E\rangle$ where V is the set of possible assignments of values to variables and E is the set of pairs of mutually consistent variable-value assignments [22]. In the *coloured microstructure*, the vertices representing an assignment to variable v_i are labelled by a colour representing variable v_i, thus maintaining the distinction between assignments to different variables.

The coloured microstructure of a CSP instance captures both the structure and the relations of a CSP instance and so it is a natural place to look for tractable classes which are neither purely structural nor purely relational.

Of the results on (coloured) microstructure properties, two are of particular note. First it was observed that the class of instances with a perfect microstructure is tractable [28]. This is a proper generalisation of the well known hybrid tractable CSP class whose instances allow arbitrary unary constraints and in which every pair of variables is constrained to be not equal [24, 29], and of the hybrid class whose microstructure is triangulated [22, 5]. The perfect microstructure property excludes an infinite set of induced subgraphs from the microstructure. In this paper, we provide a different hybrid class that also strictly generalises the class of CSP instances with an inequality constraint between every pair of variables and an arbitrary set of unary constraints, but does so by forbidding a single pattern.

Secondly, the so called broken-triangle property properly extends the structural notion of acyclicity to a more interesting hybrid class [6]. The broken triangle property is specified by excluding a particular pattern (a subgraph) in the coloured microstructure. It is this property that we generalise in this paper. We do this by working directly with the CSP instance (or equivalently its coloured microstructure) rather than its microstructure abstraction which is a simple graph. This allows us to introduce a language for expressing hybrid classes in terms of forbidden patterns, so obtaining novel hybrid tractable classes. In the case of binary negative patterns we are able to characterise very large classes of tractable and NP-hard forbidden patterns. This leaves the complexity of just one case unresolved and we conjecture that this last case is tractable, which would give us a new CSP dichotomy for hybrid classes of binary CSPs defined by negative patterns.

Contributions

In this paper we generalise the definition of a CSP instance to that of a CSP pattern which has two types of tuple in its constraint relations, tuples which are explicitly allowed/disallowed and tuples which are labelled as *unknown*[1]. By defining a natural notion of containment of patterns in a CSP, we are able to describe problems defined by *forbidden patterns*: a CSP instance P forbids a particular pattern if this pattern cannot be contained in P. By defining problems in this way, we can capture both allowed and disal-

[1]This can be viewed as the natural generalisation of the CSP to a three-valued logic.

lowed constraint tuples as well as structural properties. We use this framework to capture tractability by identifying local patterns of allowed *and* disallowed tuples (within small groups of connected constraints) whose absence is enough to guarantee tractability.

By extending the notion of the CSP instance to that of a pattern we are able to unify the following properties:

- having a particular polymorphism;
- having a hereditary (coloured) microstructure property, such as broken triangle [6]; and
- having a tree structure (tree width 1).

Using the concept of forbidden patterns, we lay foundations for a theory that can be used to reason about classes of CSPs defined by hybrid properties. Since this is the first work of this kind, we primarily focus on the simplest case: binary patterns in which tuples are either disallowed or unknown (called *negative patterns*). We give a large class of binary negative patterns which give rise to intractable classes of problems and, using this, show that any negative pattern that defines a tractable class of problems must have a certain structure. We are able to prove that this structure is nearly enough to guarantee tractability and conjecture that there is a precise condition providing dichotomy for tractability defined by forbidding binary negative patterns. Importantly, our intractability results also allow us to give a necessary condition on the form of general tractable patterns.

The remainder of the paper is structured as follows. In Section 2 we define constraint satisfaction problems, and give other definitions which will be used in the paper. Then, in Section 3, we define the notion of a CSP pattern and describe classes of problems defined by forbidden patterns. In Section 4 we show that one must take the size of patterns into account to have a notion of maximal classes defined by forbidding patterns. In Section 5 we give some examples of tractable classes defined by forbidden patterns on three variables. In general, we are not yet able to make any conjecture concerning a dichotomy for hybrid tractability defined by general forbidden patterns. However, in Section 6 we are able to give a necessary condition for such a class to be tractable. Finally, in Section 7 we summarise the results of this work and discuss directions for future research.

2 Preliminaries

Definition 2.1. *A CSP instance is a triple $\langle V, C, D\rangle$ where*

- *V is a finite set of* ***variables*** *(with $n = |V|$).*
- *D is a finite set called the* ***domain*** *(with $d = |D|$).*
- *C is a set of* ***constraints****. Each constraint $c \in C$ is a pair $c = \langle \sigma, \rho\rangle$ where*
    - *σ is a list of variables called the* ***scope*** *of c.*

- *ρ is a relation over D of arity $|\sigma|$ called the* ***relation*** *of c. It is the set of tuples allowed by c.*

A ***solution*** *to the CSP instance $P = \langle V, D, C\rangle$ is a mapping $s : V \mapsto D$ where, for each $\langle \sigma, \rho\rangle \in C$ we have $s(\sigma) \in \rho$ (where $s(\sigma)$ represents the tuple resulting from the application of s component-wise to the list of variables σ).*

The arity of a CSP is the largest arity of any of its constraint scopes. Our long-term aim is to identify all tractable subclasses of the CSP problem which can be detected in polynomial time. In this paper we describe a general theory of forbidden patterns for arbitrary arity but only consider the implications of the new theory for tractable classes of arity two (binary) problems specified by finite sets of forbidden patterns. In such cases we are certain that class membership can be decided in polynomial time.

The CSP decision problem, which asks whether a particular CSP instance has a solution, is already NP-complete for binary CSPs. For example, there is a straightforward reduction from graph colouring to this problem in which vertices i of the graph map to CSP variables v_i and edges $\{i, j\}$ map to disequality constraints $v_i \neq v_j$.

It will sometimes be convenient in this paper to use an equivalent functional formulation of a constraint. In this alternative formulation the scope σ of the constraint $\langle \sigma, \rho\rangle$ is abstracted to a set of variables and each possible assignment is seen as a function $f : \sigma \mapsto D$. The constraint relation in this alternative view is then a function from the set of possible assignments, D^σ, into the set $\{T, F\}$ where, by convention, the tuples which occur in the constraint relation are those which map to T. It follows that any assignment to the set of all variables is allowed by $\langle \sigma, \rho\rangle$ when its restriction to σ is mapped to T by ρ.

Definition 2.2. *For any function $f : X \mapsto Y$ and $S \subset X$, the notation $f|_S$ means the function with domain S satisfying $f|_S(x) = f(x)$ for all $x \in S$.*

Given a set V of variables and a domain D, a ***constraint*** *in functional representation is a pair $\langle \sigma, \rho\rangle$ where $\sigma \subseteq V$ and $\rho : D^\sigma \mapsto \{T, F\}$. A* ***CSP instance*** *in functional representation is a triple $\langle V, D, C\rangle$ where C is a set of constraints in functional representation.*

A ***solution*** *(to a CSP instance $\langle V, D, C\rangle$ in functional representation) is a mapping $s : V \mapsto D$ where, for each $\langle \sigma, \rho\rangle \in C$ we have $\rho(s|_\sigma) = T$.*

The functional formulation is clearly equivalent to the relational formulation and we will use whichever seems more appropriate throughout the paper. The choice will always be clear from the context.

Relational tractability of binary CSP

We will refer to a set of relations Γ on some finite set D as a *constraint language*.

Definition 2.3. *Let D be a finite set and let Γ be a set of relations on D. We define* $\mathrm{CSP}(\Gamma)$ *to be the set of problems for which every constraint $\langle \sigma, \rho\rangle$ satisfies $\rho \in \Gamma$.*

A constraint language Γ is said to be *tractable* if $\mathrm{CSP}(\Gamma')$ is a tractable class of problems for each finite $\Gamma' \subseteq \Gamma$. It is well-known that the tractability of Γ can be determined by studying the *polymorphisms* of Γ [4].

Definition 2.4. *Let D be a finite set and let ρ be a binary relation on D. A k-ary* ***polymorphism*** *of ρ is a function $f : D^k \mapsto D$ satisfying*

$$\forall x_1, \ldots, x_k \in \rho, \quad \langle f(x_1[1], \ldots, x_k[1]), f(x_1[2], \ldots, x_k[2])\rangle \in \rho\,.$$

It is known that the existence of a non-trivial polymorphism is a necessary condition for a set of relations to give rise to a tractable constraint language [20, 4, 1]. Using this characterisation, almost all tractable classes of the CSP defined by sets of relations have been determined, though establishing the full dichotomy still remains an open problem.

Structural tractability of binary CSP

Structural tractability considers the classes of problems defined by placing restrictions on the set of constraint scopes, but which allow arbitrary constraint relations. For simplicity, and as this is the focus of this paper, we restrict our attention to binary CSPs. In this case, the set of constraint scopes defines the *constraint graph* whose vertices are the variables and whose edges are the set of scopes of constraints whose relation is not complete (i.e. the Cartesian product D^2). All definitions and concepts extend naturally to non-binary CSPs. The key property here is the *tree width* of the constraint graph.

Definition 2.5. *Let G be a graph. A* tree decomposition *of G is a pair (T, X), where T is a tree and X is a mapping that associates with every node $t \in V(T)$ a set $X_t \subset V(G)$ such that for every $v \in V(G)$ the set $\{t \in V(T) \mid v \in X_t\}$ is connected, and for every $e \in E(G)$ there is a $t \in V(T)$ such that $e \subset X_t$.*

The width *of a tree decomposition (T, X) is $\max\{|X_t| - 1 \mid t \in V(T)\}$. The* tree width *of a graph G, denoted $tw(G)$, is the minimum of the widths of all tree decompositions of G.*

The following theorem is classical [11, 13].

Theorem 2.6. *Let P be a CSP. If the constraint graph of P, G_P, has $tw(G_P) \leq k$, then we can solve P in time $O(nd^{k+1})$.*

What is more, under reasonable technical assumptions, there is no property of G_P which gives rise to a larger tractable class of CSPs. This establishes a dichotomy for structural tractability of binary CSPs. A similar result has been obtained for CSPs of higher arity. See [9, 17, 18, 23] for more details.

3 Forbidden patterns in CSP

In this paper we explain how we can define classes of CSP instances by forbidding the occurrence of certain patterns. A CSP pattern is a generalisation of a CSP instance. In a CSP pattern we define the relations relative to a three-valued logic on $\{T, F, U\}$, meaning that the pattern is simply the set of CSP instances in which each undefined value U is replaced by either T or F. Forbidding a CSP pattern is equivalent to simultaneously forbidding all these instances as sub-problems.

Definition 3.1. *We define a three-valued logic on $\{T, F, U\}$, where U stands for* unknown *or* undefined*. The set $\{T, F, U\}$ is partially ordered so that $U < T$ and $U < F$ but T and F are incomparable.*

*Let D be a finite set. A k-ary **three-valued relation** on D is a function $\rho : D^k \mapsto \{T, F, U\}$. Given a pair of k-ary three-valued relations ρ and ρ', we say ρ **realises** ρ' if*

$$\forall x \in D^k, \rho(x) \geq \rho' \,.$$

Definition 3.2. *A **CSP pattern** is a triple $\chi = \langle V, D, C \rangle$ where:*

- *V is the set of **variables**.*
- *D is the **domain**.*
- *C is a set of constraint **patterns**. Each constraint pattern $c \in P$ is a pair $c = \langle \sigma, \rho \rangle$, where $\sigma \subseteq V$, the **scope** σ of c, is a set of variables and $\rho : D^\sigma \mapsto \{T, F, U\}$ is the **three-valued relation** (in functional representation) of c.*

*The **arity** of a CSP pattern χ is the maximum arity of any constraint pattern $\langle \sigma, \rho \rangle$ of χ.*

We will sometimes define ρ as a partial function from D^σ to $\{T, F\}$; the values for which ρ is undefined are those which are mapped to U. For simplicity of presentation, we assume throughout this paper that no two constraint patterns in C have the same scope (and that, in the case of CSP instances, that no two constraints have the same scope). We will represent binary CSP patterns by simple diagrams. Each oval represents the domain of a variable, each dot a domain value. The tuples in constraint patterns with value F are shown by dashed lines, those with value T by solid lines and those with value U are not depicted at all.

Contexts

By further adding simple structure to the domains and variable sets of patterns, we are able to make the notion of patterns more specific, and so we can capture larger, and more interesting, tractable classes. Contexts such as these have been used in the past to capture tractable classes. For example, when the domain is totally ordered we can define the tractable max-closed class [21], and when we have an independent total order for the domain of each variable we can capture the renamable Horn class [16].

The weakest such context that we will consider only allows us to say when two variables are distinct. A pattern with such a context will be called **flat**. We give a general definition of context, but in this paper the only contexts we require are total orders on the variable set or the domain.

Definition 3.3. *A* ***CSP context*** *is a set of relational structures Ω on the product of the variable set and domain*[2]*.*

A CSP pattern $\langle V, D, C\rangle$ is considered in context Ω by associating it with a structure $\omega \in \Omega$ for appropriately-sized variable set and domain.

Let $\langle V, D\rangle$ and $\langle V', D'\rangle$ be in context Ω, with ω and ω' the elements of Ω giving structure to the sets $V \times D$ and $V' \times D'$, respectively. A ***contextual homomorphism*** *is an Ω-structure preserving function $F : V \times D \mapsto V' \times D'$, i.e. for each $(u, a), (v, b) \in V \times D$, $\langle (u, a), (v, b)\rangle \in \omega$ implies $\langle F(u, a), F(v, b)\rangle \in \omega'$.*

Definition 3.4. *Two CSP patterns are* ***compatible*** *if they are considered in the same context.*

Thus, for example, two CSP patterns with totally-ordered domains are compatible even if the domain sizes are different. In this case, a contextual homomorphism between the two patterns must preserve the domain ordering.

Patterns, CSPs and occurrence

A CSP instance is just a CSP pattern in which the three-valued relations of the constraint patterns never take the value U. That is, we decide for each possible tuple whether it is in the relation or not. Furthermore, in a CSP instance, for each pair of variables we assume that a constraint exists with this scope; if no explicit constraint is given on this scope, then we assume that the relation is complete, i.e. it contains all tuples. This can be contrasted with CSP patterns for which the absence of an explicit constraint on a pair of variables implies that the truth value of each tuple is undefined.

In order to define classes of CSP instances by forbidding patterns, we require a formal definition of an occurrence (containment) of a pattern within an instance. We define the more general notion of containment of one CSP pattern within another pattern. Informally, the names of the variables and domain elements of a CSP pattern are inconsequential and a containment allows a renaming of the variables and the domain values of each variable. Thus, in order to define the containment of patterns, we firstly require a formal definition of a renaming. In an arbitrary renaming two distinct variables may map to the same variable and two distinct domain values may map to the same domain value. However, we do not allow distinct variables v_1, v_2 to map to the same variable if v_1, v_2 belong to the scope of any binary constraint (otherwise this binary constraint could not be correctly represented on a scope consisting of a single variable).

[2]We tacitly assume that a context contains at least one structure for every combination of sizes of variable set and domain.

A domain labelling of a set of variables is just an assignment of domain values to those variables. Variable and domain renaming induces a mapping on the domain labellings of scopes of constraints: we simply assign the renamed domain values to the renamed variables. There is a natural way to extend this mapping of domain labellings to a mapping of a constraint pattern: the truth-value of each mapped domain labelling is the same as the truth-value of the original domain labelling. However, it may occur that two domain labellings of some scope map to the same domain labelling, so instead the resulting value is taken to be the greatest of the original truth-values. (In order for this process to be well-defined, if two domain labellings of a constraint are mapped to the same domain labelling, then their original truth-values must be comparable.) This leads to the following formal definition of a renaming which is the first step towards the definition of containment.

Definition 3.5. *Let $\chi = \langle V, D, C\rangle$ and $\chi' = \langle V', D', C'\rangle$ be compatible CSP patterns.*

*We say that χ' is a **renaming** of χ if there exist a variable renaming function $s : V \mapsto V'$ and a domain renaming function $t : V \times D \mapsto D'$ that satisfy:*

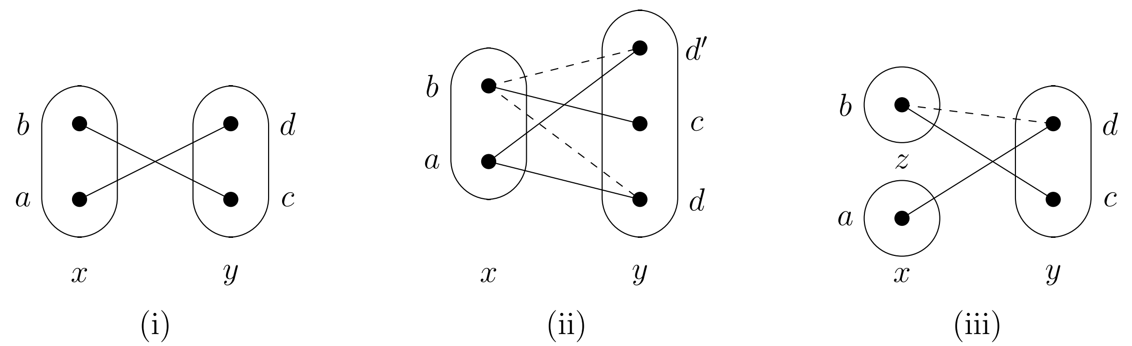
- *If $s(v_1) = s(v_2)$ for distinct variables v_1, v_2, then there is no constraint pattern $\langle\sigma, \rho\rangle \in C$ with $v_1, v_2 \in \sigma$ and ρ a non-trivial relation (a function which is not identically equal to U).*
- *$F : V \times D \mapsto V' \times D'$ defined by $F(\langle v, a\rangle) = \langle s(v), t(v, a)\rangle$ is a contextual homomorphism.*
- *For each constraint pattern $\langle\sigma, \rho\rangle \in C$, for any two domain labellings $\ell, \ell' \in D^\sigma$ for which $F(\ell) = F(\ell')$, we have that $\rho(\ell)$ and $\rho(\ell')$ are comparable.*
- *$C' = \{\langle s(\sigma), \rho'\rangle \mid \langle\sigma, \rho\rangle \in C\}$, where $\rho'(f) = \max\{\rho(\ell) \mid F(\ell) = f\}$, for each $f : s(\sigma) \mapsto D$.*

We will use patterns to define sets of CSP instances by forbidding the occurrence (containment) of the patterns in the CSP instances. In this way we will be able to characterise tractable subclasses of the CSP. Informally, a pattern χ is said to occur in a CSP instance P if we can find variables in P corresponding to the variables of χ, such that there is a constraint in P that realises each constraint pattern in χ. We will now formally define what we mean by a pattern occurring in another pattern (and by extension, in a CSP instance).

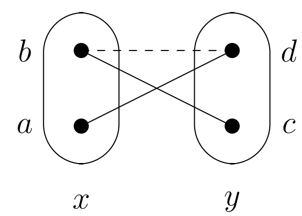
Definition 3.6. *We say that a CSP pattern χ **occurs** in a CSP pattern $P = \langle V, D, C\rangle$ (or that P **contains** χ), denoted $\chi \to P$, if there is a renaming $\langle V, D, C'\rangle$ of χ where, for every constraint pattern $\langle\sigma, \rho'\rangle \in C'$ there is a constraint pattern $\langle\sigma, \rho\rangle \in C$ and, furthermore, ρ realises ρ'.*

Example 3.7. This example describes three simple containments. Consider the three constraint patterns, Pattern 1(i)-(iii). These constraint patterns occur in, or are contained

Pattern 1



Pattern 2



in, Pattern 2 by the contextual homomorphisms F_1, F_2, and F_3, respectively, which we will now describe.

F_1 is simply a bijection. Although the patterns are different, this is a valid containment of Pattern 1(i) into Pattern 2 because the three-valued relation of Pattern 2 is a realisation of the three-valued relation in Pattern 1(i): we are replacing $(b, d) \to U$ by $(b, d) \to F$.

F_2 maps (x, a), (x, b), and (y, c) to themselves, and maps both (y, d) and (y, d') to (y, d). This merging of domain elements is possible because the three-valued constraint relation of Pattern 1(ii) agrees on tuples involving the assignments (y, d) and (y, d') and, furthermore, the restriction of the three-valued relation of Pattern 1(ii) to either of these two assignments is equivalent to the three-valued constraint relation of Pattern 2: $(b, d) \to F$ and $(a, d) \to T$.

Finally, F_3 maps (y, c) and (y, d) to themselves, and maps (x, a) and (z, b) in Pattern 1(iii) to (x, a) and (x, b), respectively, in Pattern 2. As before, this merging of variables is possible because the three-valued relations agree. □

Before continuing we need to define what we mean when we say that a class of CSP instances is definable by forbidden patterns.

Definition 3.8. *Let C be any class of CSP instances. We say that C is **definable by forbidden patterns** if there is some context Ω and some set of patterns $\mathcal{X}$ for which the set of CSP instances in which none of the patterns in $\mathcal{X}$ occur are precisely the instances in C.*

Notation: Let $\mathcal{X}$ be a set of CSP patterns with maximum arity k. We will use $\mathrm{CSP}(\overline{\mathcal{X}})$ to denote the set of CSP instances with arity at most k (compatible with $\mathcal{X}$) in which no element $\chi \in \mathcal{X}$ occurs. When $\mathcal{X}$ is a singleton $\{\chi\}$ we will allow $\mathrm{CSP}(\overline{\chi})$ to mean $\mathrm{CSP}(\overline{\{\chi\}})$.

Tractable Patterns

In this paper we will define, by forbidding certain patterns, tractable subclasses of the CSP. Furthermore, we will give examples of truly hybrid classes (i.e. not definable by purely relational or purely structural properties).

Definition 3.9. *A set of patterns $\mathcal{X}$ is* ***intractable*** *if $\mathrm{CSP}(\overline{\mathcal{X}})$ is NP-hard. It is* ***tractable*** *if there is a polynomial-time algorithm to solve $\mathrm{CSP}(\overline{\mathcal{X}})$. A single pattern χ is tractable (intractable) if $\{\chi\}$ is tractable (intractable).*

It is worth observing that classes of CSP instances defined by forbidding patterns do not have a fixed domain. Recall, however, that CSP instances have finite domains.

We will need the following simple lemma for our proofs of intractability results in later sections of this paper.

Lemma 3.10. *Let $\mathcal{X}$ and $\mathcal{T}$ be sets of compatible CSP patterns and suppose that for every pattern $\tau \in \mathcal{T}$, there is some pattern $\chi \in \mathcal{X}$ for which $\chi \rightarrow \tau$. Then $\mathrm{CSP}(\overline{\mathcal{X}}) \subseteq \mathrm{CSP}(\overline{\mathcal{T}})$.*

Proof. Let $P \in \mathrm{CSP}(\overline{\mathcal{X}})$, so $\chi \not\rightarrow P$ for each $\chi \in \mathcal{X}$. Then we cannot have $\tau \rightarrow P$ for any $\tau \in \mathcal{T}$, since this would imply that there exists some $\chi \in \mathcal{X}$ such that $\chi \rightarrow \tau \rightarrow P$ and hence that $\chi \rightarrow P$. Hence, $P \in \mathrm{CSP}(\overline{\mathcal{T}})$. $\square$

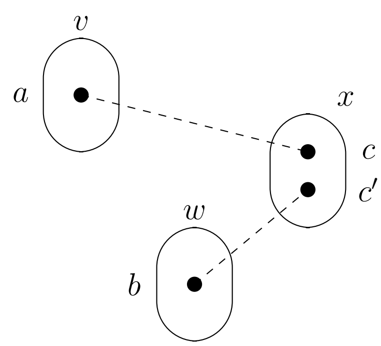
Corollary 3.11. *Let $\mathcal{X}$ and $\mathcal{T}$ be sets of compatible CSP patterns and suppose that for every pattern $\tau \in \mathcal{T}$, there is some pattern $\chi \in \mathcal{X}$ for which $\chi \rightarrow \tau$.*

We then have that $\mathrm{CSP}(\overline{\mathcal{T}})$ is intractable if $\mathrm{CSP}(\overline{\mathcal{X}})$ is intractable and conversely, that $\mathrm{CSP}(\overline{\mathcal{X}})$ is tractable whenever $\mathrm{CSP}(\overline{\mathcal{T}})$ is tractable.

Finally, we give a very simple example of a tractable pattern. This is an example of a *negative* pattern since the only truth-values in the relations are F and U. We will use the notation $\mathrm{NEQ}(v_1, \ldots, v_r)$ to denote the fact that the variables $v_1, \ldots, v_r$ are all distinct.

Example 3.12. Consider the pattern given as Pattern 3. This defines a class of CSPs which is trivially tractable. We can apply a pre-processing step consisting of first establishing arc consistency, and then assigning value c to variable x (and eliminating the variable x) if this assignment is consistent with all remaining assignments to other variables. Forbidding Pattern 3 ensures that after this pre-processing step there are no paths of length greater than 2 in the constraint graph. Thus, any problem forbidding Pattern 3 can be decomposed into a set of independent sub-problems, each of maximum size 2. $\square$

Pattern 3 A very simple negative pattern.



NEQ(v, w, x), $c \neq c'$

Relational and structural tractability as forbidden patterns

The following examples show certain strengths of this notion for defining tractable classes. First, in Example 3.13, we show that forbidden patterns properly generalise polymorphisms. Then, in Example 3.14, we show that we can define the class of tree-structured CSPs by a single forbidden pattern.

Example 3.13. Let $\langle D, <\rangle$ be any totally ordered domain. A binary relation ρ over D is said to be *max-closed* if, for any tuples $\langle a, b\rangle, \langle a', b'\rangle \in \rho$ we have that $\langle \max(a, a'), \max(b, b')\rangle \in \rho$. It is well known that the class of CSP instances whose relations are binary max-closed is tractable [21]. We can define the class of max-closed CSPs as those forbidding the following pattern (Pattern 4):

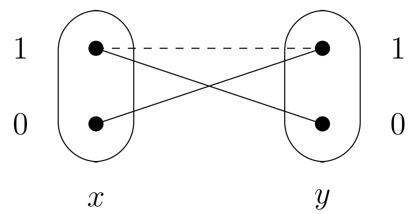
- CSP context: the variable set is unstructured and the domain is totally ordered.
- Variables: $\{x, y\}$, where $x \neq y$.
- Domain: The ordered set $\{0, 1\}$ with $0 < 1$.
- A single constraint pattern with scope $\{x, y\}$ and three-valued relation:

$$\begin{gathered}\{x \mapsto 0, y \mapsto 0\} \mapsto U \\ \{x \mapsto 0, y \mapsto 1\} \mapsto T \\ \{x \mapsto 1, y \mapsto 0\} \mapsto T \\ \{x \mapsto 1, y \mapsto 1\} \mapsto F\end{gathered}$$

In this pattern, the context specifies that $x \neq y$ and $0 < 1$, so we have limited the contextual homomorphisms to those that map x and y to distinct variables and 0 and 1 to ordered domain values. Thus, saying that a pattern MAX2 is forbidden in a CSP instance P is equivalent to saying there is no constraint allowing a pair of tuples (a, b) and (a', b'), where

$a < a'$ and $b' < b$, such that (a', b) is disallowed; this is equivalent to saying that every constraint must be max-closed. □

Pattern 4 The MAX2 pattern.



Context: $x \neq y$, $0 < 1$

The set of max-closed relations are also known as the relations which are closed under the polymorphism $\max(x, y)$.

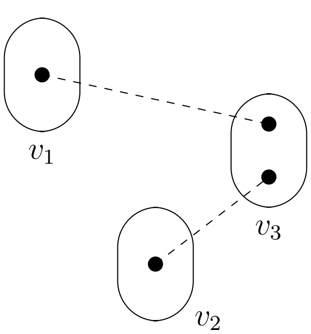
Recall from Definition 2.4 that, given a finite set D and a binary relation ρ on D, a k-ary polymorphism of ρ is a function $f : D^k \mapsto D$ satisfying

$$\forall x_1, \ldots, x_k \in \rho, \quad f(x_1, \ldots, x_k) \in \rho\,,$$

where $f(x_1, \ldots, x_k) = \langle f(x_1[1], \ldots, x_k[1]), f(x_1[2], \ldots, x_k[2]) \rangle$. Clearly, we can define the set of relations which have a particular k-ary polymorphism f as the set of relations forbidding a particular set of patterns, namely those which allow k tuples $x_1, \ldots, x_k$ but which disallow $f(x_1, \ldots, x_k)$. Thus, every class of binary CSPs defined by having a particular polymorphism can be defined using forbidden patterns.

We now turn our attention to structural tractability. In Example 3.14 below we show that a forbidden pattern can capture the class of CSPs with tree width 1.

Pattern 5 Tree structure pattern (TREE)



$v_1 < v_2 < v_3$

Example 3.14. Consider the pattern TREE, given as Pattern 5. We will show that the class CSP($\overline{\text{TREE}}$) is exactly the set of CSPs whose constraint graph is a forest (i.e. has tree width 1). First, suppose $P \in \mathrm{CSP}(\overline{\textsc{Tree}})$. Then, there exists some ordering $\pi = (v_1, \ldots, v_n)$ such that each variable shares a constraint with at most one variable preceding it in the ordering. On the other hand, suppose P is a CSP whose constraint graph is a tree. By ordering the vertices according to a pre-order traversal, we obtain an ordering in which each variable shares a constraint with at most one variable preceding it in the ordering (its parent); thus, $P \in \mathrm{CSP}(\overline{\textsc{Tree}})$.

□

4 On characterising tractable forbidden patterns

In relational tractability we can define a maximal tractable sub-problem of the CSP problem. Such a class of relations is maximal in the sense that it is not possible to add even one more relation to the set without sacrificing tractability.

In the case of structural tractability the picture is less clear, since here we measure the complexity of an infinite set of hypergraphs (or, more generally, relational structures). We obtain tractability if we have a bound on some width measure of these structures. Whatever width measure is chosen we have a containment of the class with width bounded by k inside that of the class of width bounded by $k+1$ and so no maximal class is possible. Nevertheless, for each k there is a unique maximal class of structurally tractable instances.

In this section, we show that in the case of forbidden patterns the situation is similar.

Definition 4.1. *Let $\chi = \langle V, D, C \rangle$ and $\tau = \langle V', D', C' \rangle$ be any two flat CSP patterns. We can form the disjoint unions $V \mathbin{\dot\cup} V'$ and $D \mathbin{\dot\cup} D'$. Now, extend each constraint pattern in C to be over the domain $D \mathbin{\dot\cup} D'$ by setting the value of any tuple including elements of D' to be U, and extend similarly the constraint patterns in C'. In this way we can define $C \mathbin{\dot\cup} C'$ and then we set the **disjoint union** of χ and τ to be $\chi \mathbin{\dot\cup} \tau = \langle V \mathbin{\dot\cup} V', D \mathbin{\dot\cup} D', C \mathbin{\dot\cup} C' \rangle$.*

Lemma 4.2. *Let χ and τ be flat binary CSP patterns. Then*

$$\mathrm{CSP}(\overline{\chi}) \cup \mathrm{CSP}(\overline{\tau}) \subsetneq \mathrm{CSP}(\overline{\chi \mathbin{\dot\cup} \tau})\,.$$

Moreover, we have that $\mathrm{CSP}(\overline{\chi \mathbin{\dot\cup} \tau})$ is tractable whenever $\mathrm{CSP}(\overline{\chi})$ and $\mathrm{CSP}(\overline{\tau})$ are tractable.

Proof. We begin by showing the strict inclusion

$$\mathrm{CSP}(\overline{\chi}) \cup \mathrm{CSP}(\overline{\tau}) \subsetneq \mathrm{CSP}(\overline{\chi \mathbin{\dot\cup} \tau})\,.$$

That the inclusion holds follows directly from Lemma 3.10. To see that the inclusion is strict, observe that χ and τ occur in a CSP pattern whose domain is the disjoint union of those for χ and τ but whose variable set has size equal to the larger of the two original variable sets. Any CSP instance containing this pattern is neither in $\mathrm{CSP}(\overline{\chi})$ nor in $\mathrm{CSP}(\overline{\tau})$. However, we can construct a CSP instance containing this pattern which is contained in $\mathrm{CSP}(\overline{\chi \mathbin{\dot\cup} \tau})$, as the assumption that all variables are distinct means that $\chi \mathbin{\dot\cup} \tau$ is not contained in this pattern.

Suppose $P \in \mathrm{CSP}(\overline{\chi \cup\!\!\!\cdot\, \tau})$. If $P \in \mathrm{CSP}(\overline{\chi}) \cup \mathrm{CSP}(\overline{\tau})$ then P can be solved in polynomial time, by the tractability of $\mathrm{CSP}(\overline{\chi})$ and $\mathrm{CSP}(\overline{\tau})$.

So we may suppose that $\chi \to P$. Choose a particular occurrence of χ in P and let σ denote the set of variables used in the containment. Consider any assignment $t : \sigma \mapsto D$. Let P_t denote the problem obtained by making this assignment and then enforcing arc-consistency on the resulting problem. This corresponds to adding some new unary constraints to P.

We will show that if τ occurs in P_t then $\chi \cup\!\!\!\cdot\, \tau$ must occur in P. To see this, observe that any containment of τ in P_m naturally induces a containment of τ in P that extends to a containment of $\chi \cup\!\!\!\cdot\, \tau$ in P, by considering the occurrence of χ in σ. Thus, we can conclude that $P_t \in \mathrm{CSP}(\overline{\tau})$, and so can be solved in polynomial time.

By construction, any solution to P_t extends to a solution to P by adding the assignment t to the variables σ. Moreover, every solution to P corresponds to a solution to P_t for some $t : \sigma \mapsto D$. Since the size of χ is fixed, we can iterate over the solutions to χ in polynomial time. If P has a solution, then we will find it as the solution to some P_t. If we find that no P_t has a solution, then we known P does not have a solution. Thus, since we can solve each P_t in polynomial time, we can also solve P in polynomial time. $\square$

Corollary 4.3. *No tractable class defined by forbidding a flat pattern is maximal.*

Proof. Let χ be any tractable flat pattern. Consider the pattern defined by the disjoint union of two copies of χ, which we denote $\chi^{(2)}$. By Lemma 4.2 we have that $\mathrm{CSP}(\overline{\chi^{(2)}})$ is tractable but also that

$$\mathrm{CSP}(\overline{\chi}) \subsetneq \mathrm{CSP}(\overline{\chi^{(2)}}) \,,$$

and hence $\mathrm{CSP}(\overline{\chi})$ is not a maximal tractable class. $\square$

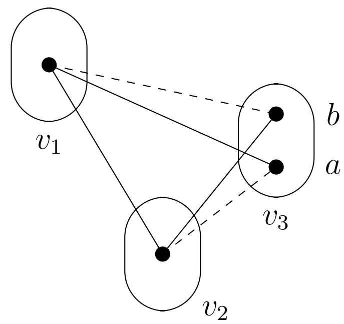
It follows that we cannot characterise tractable forbidden patterns by exhibiting all maximal classes defined by tractable forbidden patterns. Indeed, a consequence of Lemma 4.2 is that we can construct an infinite chain of patterns, such that forbidding each one gives rise to a slightly larger tractable class. Naturally, if we place an upper bound on the size of the patterns then there are only finitely many patterns that we can consider, so maximal tractable classes defined by forbidden patterns of bounded size necessarily exist.

For the moment, we are not able to make a conjecture as to the structure of a dichotomy for general forbidden patterns. Nonetheless, in Section 6, by restricting our attention to a special case, *forbidden negative patterns*, we are able to obtain interesting general results.

5 Tractable forbidden patterns on three variables

In the previous section, we showed that we need to place restrictions on the size of the forbidden patterns if we want to establish any sort of dichotomy. Since forbidden patterns on two variables only place restrictions on the set of constraint relations that can occur in an instance, the first interesting hybrid classes occur when we consider three variables. In

Pattern 6 Broken triangle pattern (BTP)



$v_1 < v_2 < v_3$

this section we present two hybrid tractable classes of binary CSP instances characterised by forbidden patterns on three variables.

The first example, already introduced in [6], is known as the **broken-triangle property** (BTP). In order to capture this class by a forbidden pattern we have to work in a context where the set of variables is totally ordered. In this case pattern containment must preserve the total order. We can define the broken-triangle property by the forbidden pattern BTP, shown in Pattern 6. The following result was proved in [6].

Theorem 5.1. *Let* BTP *be the pattern in Pattern 6. The class of CSP instances* $\mathrm{CSP}(\overline{\mathrm{BTP}})$ *can be solved in polynomial time.*

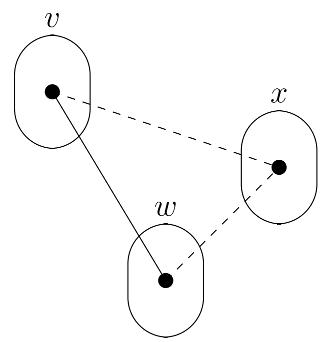
It is easy to see that TREE (shown in Pattern 5) occurs in BTP (with some truth-values U being changed to T). It follows from Lemma 3.10 that $\mathrm{CSP}(\overline{\mathrm{TREE}}) \subseteq \mathrm{CSP}(\overline{\mathrm{BTP}})$. Hence the class $\mathrm{CSP}(\overline{\mathrm{BTP}})$ includes all CSP instances whose constraint graph is a tree. However, $\mathrm{CSP}(\overline{\mathrm{BTP}})$ also includes CSP instances whose constraint graph has tree width r for any value of r: consider, for example, a CSP instance with $r+1$ variables and an identical constraint between every pair of variables which simply disallows the single tuple $\langle 0, 0\rangle$.

For any tractable forbidden pattern relative to a context with an order on the variables, we can obtain another tractable class by considering problems forbidding the pattern in a flat context. The class obtained is (possibly) smaller because it is easier to establish containment of the flat pattern. In the particular case of the broken-triangle property, we obtain a strictly smaller tractable class by forbidding Pattern 6 for all triples of variables v_1, v_2, v_3 irrespective of their order. We can easily exhibit a CSP instance that shows this inclusion to be strict: for example, the 3-variable CSP instance over Boolean domains consisting of the two constraints $v_1 = v_2$, $v_1 = v_3$ with the variable ordering $v_1 < v_2 < v_3$.

Our second example is a generalisation of the well-known tractable class of problems, ALLDIFFERENT+UNARY [24, 29]: an instance of this class consists of a set of variables V, a set of arbitrary unary constraints on V, and the constraint $v \neq w$ defined on each pair of distinct variables $v, w \in V$. We define a more general class containing every such instance using the forbidden pattern shown in Pattern 7, which we call NEGTRANS. Forbidding this pattern insists that disallowing tuples is a transitive relation, i.e. if $(\langle v, a\rangle, \langle x, b\rangle)$

and $(\langle x, b\rangle, \langle w, c\rangle)$ are disallowed then $(\langle v, a\rangle, \langle w, c\rangle)$ must also be disallowed. By the transitivity of equality, Pattern 7 does not occur in any binary CSP instance in the class AllDifferent+unary.

Pattern 7 Negative transitive pattern (Negtrans)



NEQ(v, w, x)

Theorem 5.2. *Let* Negtrans *denote Pattern 7. The class of CSP instances forbidding* Negtrans *can be solved in polynomial time.*

Proof. We prove this by a straightforward reduction to the well-known tractable problem AllDifferent+unary [8, 24].

Let $P = \langle V, D, C\rangle$ be a binary CSP in which Negtrans does not occur, and let $n = |V|$ and $d = |D|$. We define the graph G_P which we call the *inconsistency graph of P*. The vertices of G_P are the pairs $\langle v, c\rangle$ where v is a variable in P and $c \in D$ is allowed by the unary constraint on v. The edges of G_P are the pairs of vertices $\{\langle v, a\rangle, \langle w, b\rangle\}$ of G_P for which there exists a constraint $\langle\langle v, w\rangle, \rho\rangle$ with scope $\langle v, w\rangle$ such that $\langle a, b\rangle \notin \rho$. (The inconsistency graph is the microstructure complement [22] without edges between pairs of assignments to the same variable.)

We first prove that, for any connected component H of G_P, either

- The subgraph of G_P induced by H is a complete multipartite graph with edges $\{\langle v, a\rangle, \langle w, b\rangle\}$ for each $\langle v, a\rangle, \langle w, b\rangle \in H$ satisfying $v \neq w$ (in this case, we call H an *inconsistency clique*), or
- H meets exactly two variables of P: $|\{v \mid \langle v, a\rangle \in H\}| = 2$.

Any connected component H that meets only one variable is a trivial inconsistency clique. Consider a component H that meets at least three distinct variables. To show that H is an inconsistency clique we have only to show that the two end-points of every path of length three meeting three variables and of every path of length four meeting two variables are connected. (The length of a path is the number of vertices on the path).

Let $\langle v, a\rangle, \langle x, c\rangle, \langle w, b\rangle$, be a path of length three in H, where v, x, w are distinct variables. Since Negtrans does not occur in P, we must have a constraint $\langle\langle v, w\rangle, \rho\rangle$ with $\langle a, b\rangle \notin \rho$.

Let $\langle v, a\rangle, \langle w, a'\rangle, \langle v, b'\rangle, \langle w, b\rangle$ be a path of length four in H. Since H is connected and comprises at least three variables, there is some other vertex $\langle x, c\rangle$ with $x \notin \{v, w\}$ which is connected to this path. Since this creates paths of length three through three variables we can repeatedly use the argument given above to show that $\langle x, c\rangle$ is adjacent to each of the four vertices on the path. Finally, since it is adjacent to both $\langle v, a\rangle$ and $\langle w, b\rangle$ we use the argument one last time to show that these two vertices are adjacent.

We can now demonstrate the reduction to AllDifferent+unary. First we can identify all connected components of G_P in polynomial time. For each component H that is not an inconsistency clique, H meets exactly two variables v, w and there is some pair of vertices $\langle v, a\rangle$ and $\langle w, b\rangle$ which are not adjacent and which are adjacent to no vertex $\langle x, c\rangle$ for any other variable x. We can therefore make the assignments $v = a, w = b$ and remove from G_P all vertices corresponding to assignments to these variables. We denote by V' the remaining set of variables after removing each such pair of variables from P. Note that we have an assignment to every variable not in V' that is consistent with any assignment to the variables of V'.

Let $P_{V'}$ denote the resulting CSP instance on variables V' and $G_{P_{V'}}$ the corresponding inconsistency graph. The components $H_1, \ldots, H_m$ of $G_{P_{V'}}$ are all inconsistency cliques. For each component H_i and each variable v we define $H_i(v) = \{\langle w, c\rangle \in H_i \mid w = v\}$.

Consider a CSP P' with variables V' and domain $\{1, \ldots, m\}$. Apply the unary constraint on each variable v of P' given by the unary relation $\{\langle i\rangle \mid H_i(v) \neq \emptyset\}$. Finally apply the AllDifferent constraint over all variables of P'.

No solution to $P_{V'}$ can contain two assignments from the same component of $G_{P_{V'}}$. Therefore, to every solution s to $P_{V'}$ there is a corresponding solution s' to P': choose $s'(v) = i$ where $\langle v, s(v)\rangle \in H_i$.

Conversely, any solution s' to P' corresponds to a solution s to P_V by choosing $s(v)$ to be any value in $H_{s'(v)}(v)$, for each $v \in V$.

The time taken to obtain $G_{P_{V'}}$ from P is proportional to the total number of disallowed tuples in P; hence, the time taken is $O(|C|d^2)$. Solving P' is equivalent to finding a perfect matching in a bipartite graph with $|V'| + m$ vertices and up to $|V'|m$ edges. Using the *Fibonacci heap* data structure, we can find a perfect matching in a bipartite graph with N vertices and M edges in time $O(N^2 \log(N) + NM)$ [12]. Thus, we can find a solution to P' in time $O((n+m)^2 \log(n+m) + (n+m)nm)$. The maximum value of m occurs when each component of G_P contains exactly three assignments, so we will always have $m \leq \frac{nd}{3}$. Thus, under the reasonable assumption that $d \leq n$, we can solve P' in time $O(n^3d^2)$. Since $|C|$ is $O(n^2)$, it follows that P can be solved in time $O(n^3d^2)$. □

It has recently been shown [7] that the tractable class defined by forbidding Pattern 7 (Negtrans) can be extended to soft constraint problems but that this is not the case for the class of problems obtained by forbidding Pattern 6 (btp) (in the sense that the class becomes NP-hard if all unary soft constraints are also allowed).

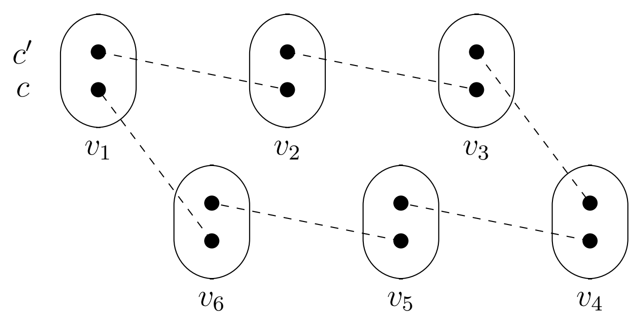
Having demonstrated through the btp and Negtrans patterns that forbidding patterns provides a language which is rich enough to define interesting hybrid tractable classes, we concentrate in the rest of the paper on progress towards characterising tractable forbidden patterns.

6 Binary flat negative patterns

In this section we define three particular patterns and one infinite class of patterns. We then use these patterns to characterise a very large class of intractable patterns. We prove that any finite set of patterns not in this class has a simple structure: one of the patterns must contain one of a particular set of patterns, which we call *pivots*. This means that any tractable finite set of patterns must include a pattern in which a pivot pattern occurs. Furthermore, we exhibit a class of patterns which are contained in pivots and which we are able to prove give rise to a tractable class. We conjecture that pivots are also tractable; if this is the case then it implies a simple characterisation of the tractability of finite sets of binary flat *negative* patterns.

Definition 6.1. *A constraint pattern $\langle\sigma, \rho\rangle$ will be called* ***negative*** *if ρ never takes the value T. A CSP pattern χ is negative if every constraint pattern in χ is negative.*

Pattern 8 CYCLE(6)



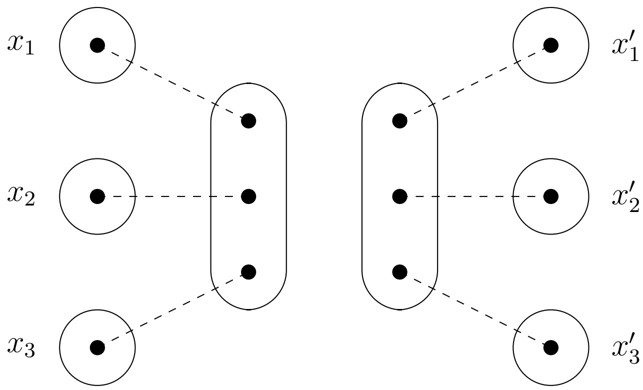
Context: $NEQ(v_1, \ldots, v_6)$

In Definition 6.2 below, we define the concept of a connected negative binary pattern. These correspond to negative binary patterns χ such that every realisation of χ as a binary CSP instance has a connected constraint graph. We first generalise the notion of constraint graph to CSP patterns. We call the resulting graph the negative structure graph.

Definition 6.2. *Let χ be any binary negative pattern. The vertices of the* ***negative structure graph*** *G are the variables of χ. A pair of vertices is connected in G if and only if they form a scope in χ whose constraint pattern assigns at least one tuple the value F. We say that χ is* ***connected*** *if its negative structure graph is connected.*

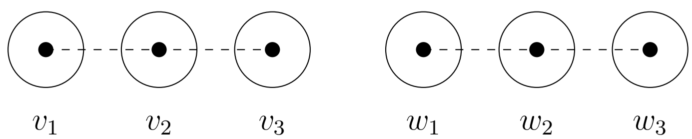
For example, Pattern 9 (VALENCY), Pattern 10 (PATH) and Pattern 11 (VALENCY+PATH) are not connected. Note that a pattern which is not connected may occur in a connected pattern (and vice versa). Pattern 8 shows CYCLE(6) which is connected. This is just one

Pattern 9 VALENCY



Context: $NEQ(x_1, x_2, x_3, x'_1) \wedge NEQ(x'_1, x'_2, x'_3)$

Pattern 10 PATH



$NEQ(v_1, v_2, v_3, w_1) \wedge NEQ(w_1, w_2, w_3)$

example of the generic pattern CYCLE(k) where $k \geq 2$. The only context for CYCLE(k) is that all variables are distinct, except for the special case $k = 2$ for which the context also includes $c \neq c'$, meaning that CYCLE(2) is composed of a single binary constraint pattern containing two *distinct* inconsistent tuples. The following theorem uses these patterns to show that most patterns are intractable.

Theorem 6.3. *Let $\mathcal{X}$ be any finite set of connected negative binary patterns. If at least one of* CYCLE(k) *(for some $k \geq 2$),* VALENCY*,* PATH*, or* VALENCY+PATH *occurs in each $\chi \in \mathcal{X}$ then* CSP($\overline{\mathcal{X}}$) *is intractable.*

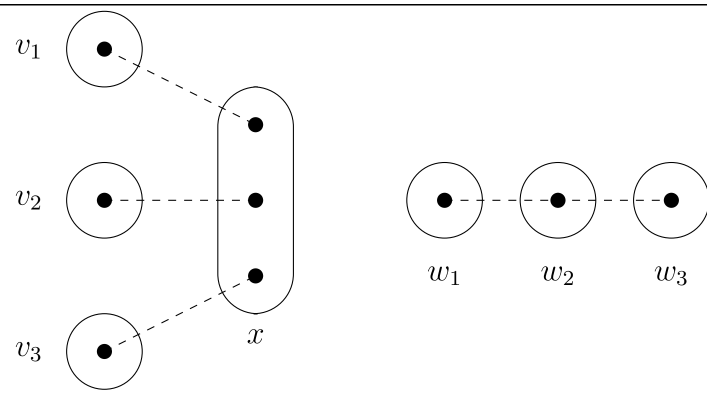
Proof. Let $\mathcal{X}$ be a set of connected negative binary patterns and let ℓ be the number of variables in the largest member of $\mathcal{X}$.

Assuming at least one of the four patterns occurs in each $\chi \in \mathcal{X}$, we can construct a class of CSPs in which no element of $\mathcal{X}$ occurs and to which we have a polynomial reduction from the well-known NP-complete problem 3SAT [14].

The construction will involve three gadgets, examples of which are shown in Figure 1. These gadgets each serve a particular purpose:

1. The *cycle gadget*, given in Figure 1(a) for the special case of 4 variables, enforces that a cycle of Boolean variables $(v_1, v_2, \ldots, v_r)$ all take the same value.

Pattern 11 VALENCY+PATH



$NEQ(v_1, v_2, v_3)$, $NEQ(w_1, w_2, w_3)$, and $x \neq w_2$

2. The *clause gadget* in Figure 1(b) is equivalent to the clause $v_1 \vee v_2 \vee v_3$, since v_C has a value in its domain if and only if one of the three v_i variables is set to true. We can obtain all other 3-clauses on these three variables by inverting the domains of the v_i variables.

3. The *line gadget* in Figure 1(c), imposes the constraint $v_1 \Rightarrow v_2$. It can also be used to impose the logically equivalent constraint $\neg v_2 \Rightarrow \neg v_1$.

Now, suppose that we have an instance of 3SAT with n propositional variables $X_1, \ldots, X_n$ and m clauses $C_1, \ldots, C_m$.

We begin to build a CSP instance Ψ to solve this 3SAT instance by using n copies of the cycle gadget (Figure 1(a)), each with $m(\ell+1)$ variables. For $i = 1, \ldots, n$, the variables along the ith copy of this cycle are denoted by $(v_i^1, v_i^2, \ldots, v_i^{m(\ell+1)})$. In any solution to a CSP instance P_Ψ with these and other constraints, we will have that the variables $v_i^j, j = 1, \ldots, m(\ell+1)$ must all have the same value, d_i. We can therefore consider each v_i^j as a copy of X_i.

Consider the clause C_w. There are eight cases to consider but they are all very similar so we will show the details for just one case. Suppose that $C_w \equiv X_i \vee X_j \vee \neg X_k$. We build the clause gadget (Figure 1(b)) with the three Boolean variables being c_w^i, c_w^j and c_w^k and invert the domain of c_w^k since it occurs negatively in C_w. Then any solution s to our constructed CSP must satisfy $s(c_w^i) \vee s(c_w^j) \vee \neg s(c_w^k) = T$.

We complete the insertion of C_w into the CSP instance by adding some length $\ell+1$ line constructions (Figure 1(c)). We connect the cycle gadgets corresponding to X_i, X_j and X_k to the clause gadget for clause C_w since X_i, X_j and X_k occur in C_w. We connect $v_i^{w(\ell+1)}$ to c_w^i since X_i is positive in C_w, so $s(c_w^i) = T$ is only possible when $s(v_i^{w(\ell+1)}) = T$, for any solution s. Similarly, we connect $v_j^{w(\ell+1)}$ to c_w^j. Finally, since X_k occurs negatively in C_w, we impose the line constraints in the other direction. This ensures that $s(c_w^k) = F$ is only possible when $s(v_k^{w(\ell+1)}) = F$. Imposing these constraints ensures that a solution is only possible when at least one of the cycles corresponding to variables X_i, X_j, and X_k is

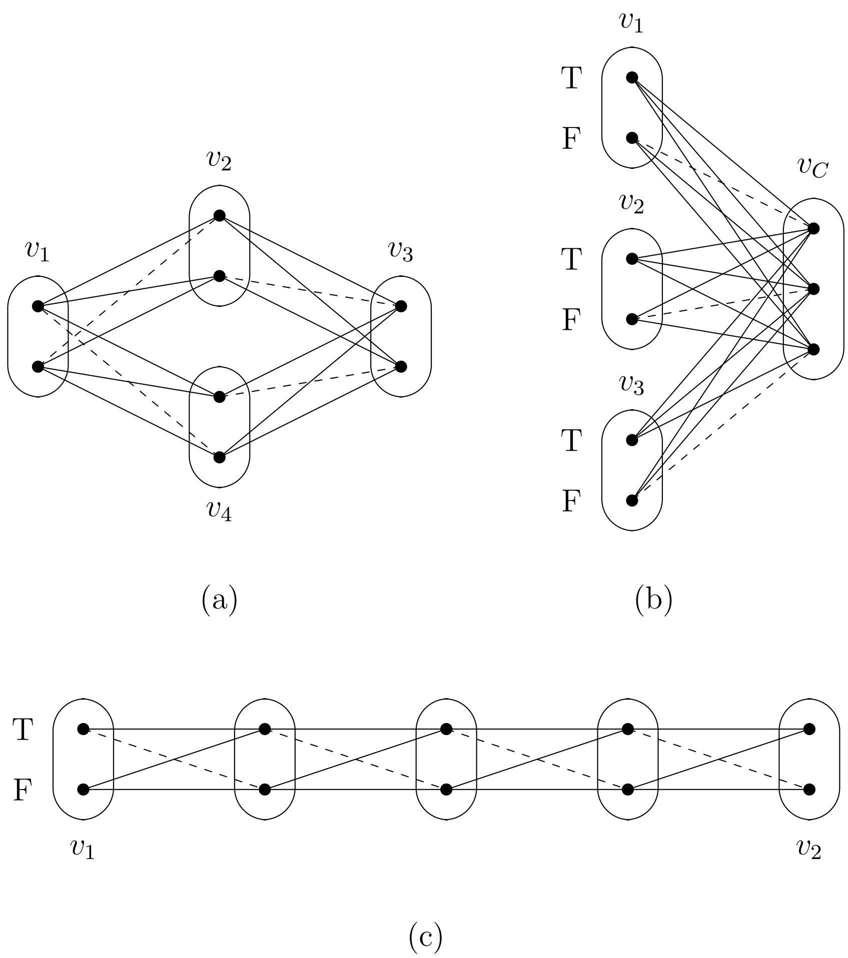


Figure 1: (a) Making copies of the same variable ($v_1 = v_2 = v_3 = v_4$). (b) Imposing the ternary constraint $v_C = v_1 \vee v_2 \vee v_3$. (c) A line of constraints which imposes $v_1 \Rightarrow v_2$.

assigned a value that would make the corresponding literal in C_w true.

We continue this construction for each clause of the 3SAT instance. Since ℓ is a constant, this is clearly a polynomial reduction from 3SAT.

We now show that any CSP instance P_Ψ constructed in the manner we have just described cannot contain any pattern in $\mathcal{X}$. We do this by showing that no pattern containing $\textsc{Cycle}$(k) (for $2 \leq k \leq \ell$), $\textsc{Valency}$, $\textsc{Path}$, or $\textsc{Valency+Path}$ can occur in the instance. This is sufficient to show that the CSP P_Ψ does not contain any of the patterns in $\mathcal{X}$.

In the CSP P_Ψ no constraint contains more than one inconsistent tuple. Thus, any $\chi \in \mathcal{X}$ for which $\textsc{Cycle}(2) \rightarrow \chi$ cannot occur in P_Ψ. Furthermore, P_Ψ is built from cycles of length $m(\ell+1)$ and paths of length $\ell+1$, and so cannot contain any cycles on less than $\ell+1$ vertices. Thus, since ℓ is the maximum number of vertices in any component of $\mathcal{X}$, it follows that no $\chi \in \mathcal{X}$ for which $\textsc{Cycle}(k) \rightarrow \chi$, for any $k \geq 3$, can occur in P_Ψ.

We define the *valency* of a variable x to be the number of distinct variables which share a constraint with x. Suppose $\textsc{Valency} \rightarrow \chi$. For this to be possible we require that there is a variable of valency four in χ, or a pair of variables of valency three connected by a path of length at most ℓ. Certainly P_Ψ has no variables of valency four. Moreover, the fact that P_Ψ was built using paths of length $\ell+1$ means that no two of its valency three variables are joined by a path of length at most ℓ. Thus, any $\chi \in \mathcal{X}$ for which $\textsc{Valency} \rightarrow \chi$ does not occur in P_Ψ.

Next, consider that case when $\textsc{Path} \rightarrow \chi$. Here χ must have two distinct (but possibly overlapping) three-variable lines (with inconsistent tuples in these constraints that match at domain values) separated by at most ℓ variables. The only place where inconsistent tuples can meet in P_Ψ is when we connect the line gadget to the cycle gadget. These connection sites are always at distance greater than ℓ, so we can conclude that $\chi \not\rightarrow P_\psi$ whenever $\textsc{Path} \rightarrow \chi$.

Finally, consider the case where $\textsc{Valency+Path} \rightarrow \chi$. Here, χ must have one variable of valency at least 3 and a path of constraints on three variables with intersecting negative tuples, and these must be connected by a path of less than ℓ variables. As observed above, the only places where we can have inconsistent tuples meeting is where the line gadget meets the cycle gadget, and there is a path of at least ℓ variables between each one of these points and every other variable of valency 3. Thus, $\chi \not\rightarrow P_\psi$ whenever $\textsc{Valency+Path} \rightarrow \chi$. $\square$

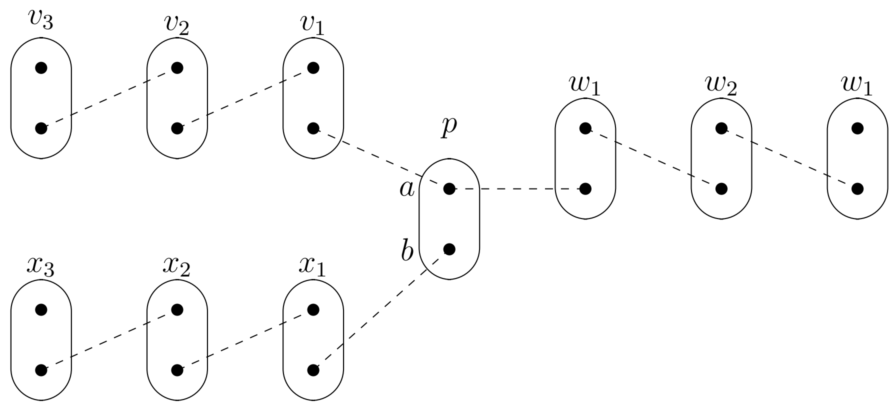
It remains to consider which sets of negative binary patterns could be tractable. For this, we need to define the *pivot* patterns, $\textsc{Pivot}(r)$, which contain every tractable pattern.

Definition 6.4. *Let* $V = \{p\} \cup \{v_1, \ldots, v_r\} \cup \{w_1, \ldots, w_r\} \cup \{x_1, \ldots, x_r\}$ *and let* $D = \{a, b\}$. *We define the pattern* $\textsc{Pivot}(r) = (V, D, C_p \cup C_v \cup C_w \cup C_x)$*, where*

$$\begin{aligned}
C_p &= \{\{(\langle p,a\rangle, \langle v_1,b\rangle)\} \mapsto F, \{(\langle p,a\rangle, \langle w_1,b\rangle)\} \mapsto F, \{(\langle p,b\rangle, \langle x_1,b\rangle)\} \mapsto F\} \\
C_v &= \{\{(\langle v_i,a\rangle, \langle v_{i+1},b\rangle)\} \mapsto F \mid i = 1, \ldots, r-1\} \\
C_w &= \{\{(\langle w_i,a\rangle, \langle w_{i+1},b\rangle)\} \mapsto F \mid i = 1, \ldots, r-1\} \\
C_x &= \{\{(\langle x_i,a\rangle, \langle x_{i+1},b\rangle)\} \mapsto F \mid i = 1, \ldots, r-1\}
\end{aligned}$$

See Pattern 12 for an example, $\textsc{Pivot}(3)$.

Pattern 12 PIVOT(3)



$$NEQ(p, v_1, v_2, v_3, w_1, w_2, w_3, x_1, x_2, x_3)$$

The following proposition characterises those sets of binary flat negative patterns which Theorem 6.3 does not prove intractable.

Proposition 6.5. *Any connected binary flat negative pattern χ either contains* CYCLE(k) *(for some $k \geq 2$),* VALENCY, PATH, *or* VALENCY+PATH, *or it itself occurs in* PIVOT(r) *for some integer $r \leq |\chi|$.*

Proof. Suppose χ does not contain any of the patterns VALENCY, CYCLE(k) (for any $k \geq 2$), PATH, or VALENCY+PATH. Since CYCLE(2) $\not\rightarrow \chi$, each constraint pattern in χ contains at most one inconsistent tuple. Recall that the *valency* of a variable x is the number of distinct variables which share a constraint with x. Since χ does not contain VALENCY it can only contain one variable of valency three and all other variables must have valency at most two. Moreover, since CYCLE(k) $\not\rightarrow \chi$ for $k \geq 3$, the negative structure graph of χ does not contain any cycles. Then, since χ is connected, the negative structure graph of χ consists of up to three disjoint paths joined at a single vertex. The fact that χ does not contain PATH means there can be at most one pair of intersecting inconsistent tuples in χ and, moreover, the fact that χ does not contain VALENCY+PATH means that this intersection must occur on the variable with valency 3, if it exists. It then follows that χ must occur in PIVOT(r), for some $r \leq |\chi|$. □

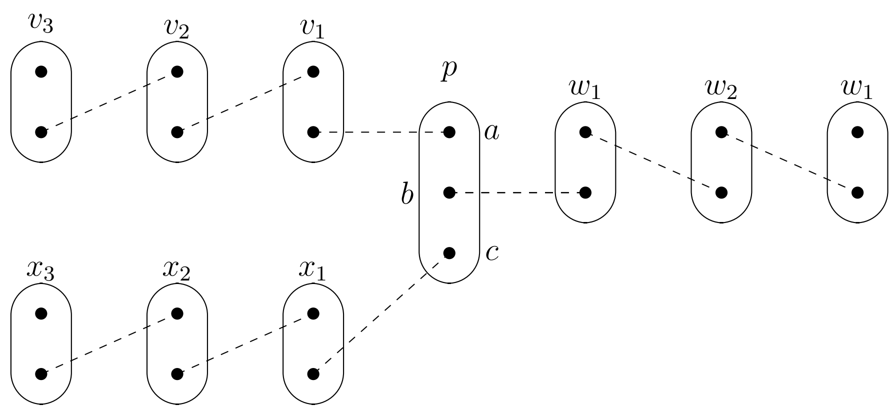
Corollary 6.6. *Let $\mathcal{X}$ be a set of connected binary flat negative patterns. Then* CSP($\overline{\mathcal{X}}$) *is tractable only if there is some $\chi \in \mathcal{X}$ that occurs in* PIVOT(r)*, for some integer $r \leq |\chi|$.*

For an arbitrary (not necessarily flat or negative) binary CSP pattern χ, we denote by NEG(χ) the flat negative pattern obtained from χ by replacing all truth-values T by U in all constraint patterns in χ and ignoring the context. For a set of patterns $\mathcal{X}$, NEG($\mathcal{X}$)

is naturally defined as the set $\textsc{neg}(\mathcal{X}) = \{\textsc{neg}(\chi) : \chi \in \mathcal{X}\}$. Clearly $\mathrm{CSP}(\overline{\textsc{neg}(\mathcal{X})}) \subseteq \mathrm{CSP}(\overline{\mathcal{X}})$. The following result follows immediately from Corollary 6.6.

Corollary 6.7. *Let $\mathcal{X}$ be a set of binary patterns such that for each $\chi \in \mathcal{X}$, $\textsc{neg}(\chi)$ is connected. Then $\mathrm{CSP}(\overline{\mathcal{X}})$ is tractable only if there is some $\chi \in \mathcal{X}$ such that $\textsc{neg}(\chi)$ occurs in $\textsc{Pivot}(r)$, for some integer $r \leq |\chi|$.*

Pattern 13 $\textsc{SepPivot}(3)$



$$NEQ(p, v_1, v_2, v_3, w_1, w_2, w_3, x_1, x_2, x_3)$$

We now define a pattern we call a *separable pivot*; forbidding this pattern defines a subclass of $\mathrm{CSP}(\overline{\textsc{Pivot}(r)})$.

Definition 6.8. *Let $V = \{p\} \cup \{v_1, \ldots, v_r\} \cup \{w_1, \ldots, w_r\} \cup \{x_1, \ldots, x_r\}$ and let $D = \{a, b, c\}$. We define the pattern $\textsc{SepPivot}(r) = (V, D, C_p \cup C_v \cup C_w \cup C_x)$, where*

$$\begin{aligned}
C_p &= \{\{(\langle p, a\rangle, \langle v_1, b\rangle)\} \mapsto F, \{(\langle p, b\rangle, \langle w_1, b\rangle)\} \mapsto F, \{(\langle p, c\rangle, \langle x_1, b\rangle)\} \mapsto F\} \\
C_v &= \{\{(\langle v_i, a\rangle, \langle v_{i+1}, b\rangle)\} \mapsto F \mid i = 1, \ldots, r-1\} \\
C_w &= \{\{(\langle w_i, a\rangle, \langle w_{i+1}, b\rangle)\} \mapsto F \mid i = 1, \ldots, r-1\} \\
C_x &= \{\{(\langle x_i, a\rangle, \langle x_{i+1}, b\rangle)\} \mapsto F \mid i = 1, \ldots, r-1\}
\end{aligned}$$

See Pattern 13 for an example, $\textsc{SepPivot}(3)$.

Clearly, $\textsc{SepPivot}(r)$ occurs in $\textsc{Pivot}(r)$: we take a bijection between corresponding variable-value pairs for the v_i, w_i and x_i variables, map both $\langle p, a\rangle$ and $\langle p, b\rangle$ in $\textsc{SepPivot}(r)$ to $\langle p, a\rangle$ in $\textsc{Pivot}(r)$, and map $\langle p, c\rangle$ in $\textsc{SepPivot}(r)$ to $\langle p, b\rangle$ in $\textsc{Pivot}(r)$. We will now show that $\mathrm{CSP}(\overline{\textsc{SepPivot}(r)})$ is tractable for any fixed r.

Theorem 6.9. *Let r be a fixed integer. $\mathrm{CSP}(\overline{\textsc{SepPivot}(r)})$ can be solved in polynomial time.*

Proof. By the grid minor theorem of Robertson and Seymour [26], there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that any graph G with tree width $tw(G) \geq f(r)$ must contain an $r \times r$ grid as a minor.

Now $\textsc{SepPivot}(r)$ occurs in any CSP instance whose constraint graph contains a vertex that starts three disjoint paths. Certainly, any CSP instance P whose constraint graph contains an $r \times r$ grid as a minor will satisfy this condition. Hence, $P \in \text{CSP}(\overline{\textsc{SepPivot}(r)})$ is only possible when $tw(P) < f(r)$. Since the class of CSP instances with tree width bounded above by $f(r)$ is tractable, it follows that $\text{CSP}(\overline{\textsc{SepPivot}(r)})$ is tractable.[3]. $\square$

The following corollary is a direct consequence of Lemma 4.2.

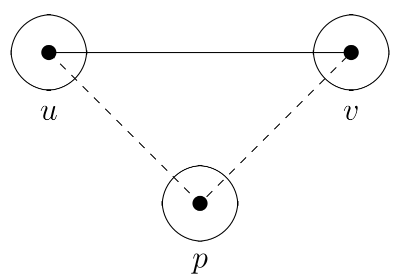
Corollary 6.10. *Any disjoint union of* $\textsc{SepPivot}(r)$ *patterns is a tractable pattern.*

Next, we show that forbidding $\textsc{Pivot}(1)$ gives rise to a tractable class of CSPs.

Proposition 6.11. $\text{CSP}(\overline{\textsc{Pivot}(1)})$ *can be solved in polynomial time.*

Proof. We will show that every $P \in \text{CSP}(\overline{\textsc{Pivot}(1)})$ can be reduced in polynomial time to $P' \in \text{CSP}(\overline{\textsc{Negtrans}})$ such that P has a solution if and only if P' has a solution. Without loss of generality, we assume that P is arc-consistent since eliminating domain values (by arc consistency) cannot destroy the fact that P is $\textsc{Pivot}(1)$-free.

Suppose $\textsc{Negtrans}$ occurs in P at $\{u, p, v\}$:



Since P does not contain $\textsc{Pivot}(1)$, it follows that p cannot be connected to any variables other than u, v in the constraint graph of P. Thus, we can obtain an equivalent CSP P_1 by eliminating p and tightening the constraint on $\{u, v\}$ by disallowing any assignment which does not extend to an assignment of p. This new CSP is still $\textsc{Pivot}(1)$-free but has had the occurrence of $\textsc{Negtrans}$ on $\{u, p, v\}$ eliminated. To see that P_1 is $\textsc{Pivot}(1)$-free, suppose that the pair of assignments $(\langle u, a\rangle, \langle v, b\rangle)$ becomes incompatible in P_1 after elimination of variable p from P. By arc consistency of P, there are values c, d such that the pairs $(\langle u, a\rangle, \langle p, c\rangle)$, $(\langle v, b\rangle, \langle p, d\rangle)$ are consistent in P. But, since $(\langle u, a\rangle, \langle v, b\rangle)$ cannot be extended to an assignment of p in P, this implies that the pairs $(\langle u, a\rangle, \langle p, d\rangle)$, $(\langle v, b\rangle, \langle p, c\rangle)$ are necessarily inconsistent in P. Now, if the inconsistent pair $(\langle u, a\rangle, \langle v, b\rangle)$ were part of an occurrence of $\textsc{Pivot}(1)$ in P_1, then we could easily obtain an occurrence of $\textsc{Pivot}(1)$

[3]The best upper bound on the function $f(r)$ is superexponential: 20^{2r^5} [25]. Thus, Theorem 6.9 does not actually provide a practical algorithm for solving problems in $\text{CSP}(\overline{\textsc{SepPivot}(r)})$.

in P by replacing $(\langle u,a\rangle, \langle v,b\rangle)$ by either $(\langle u,a\rangle, \langle p,d\rangle)$ or $(\langle v,b\rangle, \langle p,c\rangle)$ (depending on whether it is variable u or v which is at the centre of the pivot).

Thus, by repeatedly identifying and eliminating occurrences of NEGTRANS, we will eventually (after the elimination of at most $n-2$ variables) obtain a CSP $P' \in \mathrm{CSP}(\overline{\textsc{Negtrans}})$. By the way we have constructed P', we know that any solution to P' can be extended to an assignment on the removed variables. Thus, since we can solve any instance $P' \in \mathrm{CSP}(\overline{\textsc{Negtrans}})$ in polynomial time (Theorem 5.2), we can solve any instance P from $\mathrm{CSP}(\overline{\textsc{Pivot}(1)})$ in polynomial time. □

Proposition 6.11 is important as it gives us a tractable class of CSPs defined by forbidding a negative pattern which, unlike $\mathrm{CSP}(\overline{\textsc{SepPivot}(r)})$, contains problems of unbounded tree width, and so cannot be captured by structural tractability. As an example of a class of CSP instances in $\mathrm{CSP}(\overline{\textsc{Pivot}(1)})$ with unbounded tree width, consider the n-variable CSP instance P_n with domain $\{1,\ldots,n\}$ whose constraint graph is the complete graph and, for each pair of distinct values $i,j \in \{1,\ldots,n\}$, the constraint on variables v_i, v_j disallows a single pair of assignments $(\langle v_i,j\rangle, \langle v_j,i\rangle)$. Since each assignment $\langle v_i,j\rangle$ occurs in a single inconsistent tuple, PIVOT(1) does not occur in P_n, and hence $P_n \in \mathrm{CSP}(\overline{\textsc{Pivot}(1)})$.

We conjecture that there exists a larger class of tractable problems defined by forbidding negative patterns.

Conjecture 6.12. *For a fixed integer r, $\mathrm{CSP}(\overline{\textsc{Pivot}(r)})$ can be solved in polynomial time.*

A positive answer to Conjecture 6.12, taken in conjunction with Corollary 6.6, would give a dichotomy result for CSPs defined by forbidding a finite set of binary flat negative patterns, which we state in the form of a conjecture.

Conjecture 6.13. *Let $\mathcal{X}$ be a finite set of connected binary flat negative patterns. Then $\mathrm{CSP}(\overline{\mathcal{X}})$ is tractable if and only if there is some $\chi \in \mathcal{X}$ that is contained in* PIVOT(r)*, for some integer $r \leq |\chi|$.*

7 Conclusion

In this paper we described a framework for identifying classes of CSPs in terms of *forbidden patterns*, to be used as a tool for identifying tractable classes of the CSP. We gave several examples of small patterns that can be used to define tractable classes of CSPs.

In the search for a general result, we restricted ourselves to the special case of binary flat negative patterns. In Theorem 6.3 we showed that $\mathrm{CSP}(\overline{\mathcal{X}})$ is NP-hard if every pattern in a set $\mathcal{X}$ contains at least one of four patterns (Patterns 8, 9, 10, and 11). Moreover, we showed that any connected binary flat negative pattern that did not contain any of these patterns must itself be contained within a special type of pattern called a *pivot*. Hence, the presence of a pivot is a necessary condition for pattern χ to be tractable. We were able to show that another pattern, which we call *separable pivot*, occurs in the pivot pattern and defines a tractable class. Hence, separable pivots define a tractable subclass of the class defined by pivots. We conjecture that tractability extends to the whole class of problems defined by pivots. We proved tractability for pivots of size 1.

The main open problem is the resolution of the tractability of pivots of any fixed size r. Beyond this, it will be interesting to see what new tractable classes can be defined by more general binary patterns or by non-binary patterns. In particular, an important area of future research is determining all maximal tractable classes of problems defined by patterns of some fixed size (given by the number of variables or the number of variable-value assignments).

References

[1] Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.

[2] Andrei A. Bulatov. Tractable conservative constraint satisfaction problems. In *LICS 2003: Proceedings of 18th IEEE Symposium on Logic in Computer Science*, pages 321–330, 22–25 jun 2003. Available from: http://csdl.computer.org/comp/proceedings/lics/2003/1884/00/18840321abs.htm, doi:10.1109/LICS.2003.1210072.

[3] Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006. doi:10.1145/1120582.1120584.

[4] David Cohen and Peter Jeavons. The complexity of constraint languages. In Rossi et al. [27], chapter 8, pages 245–280.

[5] David A. Cohen. A new class of binary CSPs for which arc-consistency is a decision procedure. In *CP 2003: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, number 2833 in Lecture Notes in Computer Science, pages 807–811. Springer-Verlag, 2003. doi:10.1007/978-3-540-45193-8_57.

[6] Martin C. Cooper, Peter G. Jeavons, and András Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artificial Intelligence*, 174(9–10):570–584, June 2010. doi:10.1016/j.artint.2010.03.002.

[7] Martin C. Cooper and Stanislav Živný. Hybrid tractability of soft constraint problems. arXiv:1008.4071, aug 2010. Available from: http://arxiv.org/abs/1008.4071.

[8] Marie-Christine Costa. Persistency in maximum cardinality bipartite matchings. *Operations Research Letters*, 15(3):143–149, 1994. doi:10.1016/0167-6377(94)90049-3.

[9] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP 2002: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, number 2470 in Lecture Notes in Computer Science, pages 310–326. Springer-Verlag, 2002. doi:10.1007/3-540-46135-3_21.

[10] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987. doi:10.1016/0004-3702(87)90002-6.

[11] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989. doi:10.1016/0004-3702(89)90037-4.

[12] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, July 1987. doi:10.1145/28869.28874.

[13] Eugene C. Freuder. Complexity of K-Tree Structured Constraint Satisfaction Problems. In *AAAI '90: Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 4–9, 1990. Available from: http://www.aaai.org/Library/AAAI/1990/aaai90-001.php.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA., 1979.

[15] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002. doi:10.1006/jcss.2001.1809.

[16] Martin J. Green and David A. Cohen. Tractability by approximating constraint languages. In *CP 2003: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, volume 2833 of *Lecture Notes in Computer Science*, pages 392–406. Springer-Verlag, 2003. doi:10.1007/978-3-540-45193-8_27.

[17] Martin Grohe. The structure of tractable constraint satisfaction problems. In *MFCS '06: Proceedings of the 31st Symposium on Mathematical Foundations of Computer Science*, volume 4162 of *Lecture Notes in Computer Science*, pages 58–72. Springer-Verlag, 2006. doi:10.1007/11821069_5.

[18] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1–24, 2007. doi:10.1145/1206035.1206036.

[19] Marc Gyssens, Peter G. Jeavons, and David A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66(1):57–89, 1994. doi:10.1016/0004-3702(94)90003-5.

[20] Peter Jeavons, David Cohen, and Marc Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997. doi:10.1145/263867.263489.

[21] Peter G. Jeavons and Martin C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995. doi:10.1016/0004-3702(95)00107-7.

[22] P. Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *AAAI '93: Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 731–736, 1993. Available from: http://www.aaai.org/Library/AAAI/1993/aaai93-109.php.

[23] Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*, pages 735–744. ACM, 2010. doi:10.1145/1806689.1806790.

[24] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI '94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 1, pages 362–367, 1994. Available from: http://www.aaai.org/Library/AAAI/1994/aaai94-055.php.

[25] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994. doi:10.1006/jctb.1994.1073.

[26] Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41:92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

[27] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier, 2006.

[28] András Z. Salamon and Peter G. Jeavons. Perfect constraints are tractable. In *CP 2008*, volume 5202 of *Lecture Notes in Computer Science*, pages 524–528. Springer-Verlag, 2008. doi:10.1007/978-3-540-85958-1_35.

[29] W. J. van Hoeve. The alldifferent constraint: A survey. In *Proceedings of the 6th Annual Workshop of the ERCIM Working Group on Constraints*, 2001. Available from: http://arxiv.org/abs/cs/0105015v1.